Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/DK05/000090

International filing date: 10 February 2005 (10.02.2005)

Document type: Certified copy of priority document

Document details: Country/Office: US
Number: 60/542,861
Filing date: 10 February 2004 (10.02.2004)

Date of receipt at the International Bureau: 29 March 2005 (29.03.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)

World Intellectual Property Organization (WIPO) - Geneva, Switzerland
Organisation Mondiale de la Propriété Intellectuelle (OMPI) - Genève, Suisse

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

March 02, 2005

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE UNDER 35 USC 111.

APPLICATION NUMBER: *60/542,861*
FILING DATE: *February 10, 2004*

DK/05/90

By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

M. Sias

M. SIAS
Certifying Officer

13281 U.S.PTO

Please type a plus sign (+) inside this box → +

PTO/SB/16 (10-01)
Approved for use through 10/31/2002. OMB 0651-0032
Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

22154 U.S. PTO 60/542861 021004

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a requ st for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53 (c).

Express Mail Label No.

INVENTOR(S)

Given Name (first and middle [if any])	Family Name or Surname	Residence (City and either State or Foreign Country)
HANS MARTIN BOESGAARD	SORENSEN	Lyngby, Denmark

☐ *Additional inventors are being named on the _____ separately numbered sheets attached hereto*

TITLE OF THE INVENTION (280 characters max)

XMAC - A NEW FAST PROVABLE SECURE MAC

CORRESPONDENCE ADDRESS

Direct all correspondence to:

☒ Customer Number 30593 → 30593

OR *Type Customer Number here*

☐ Firm *or* Individual Name	HARNESS, DICKEY & PIERCE, P.L.C.				
Address	P.O. Box 8910				
Address					
City	Reston	State	VA	ZIP	20195
Country	USA	Telephone	703-668-8000	Fax	703-668-8200

ENCLOSED APPLICATION PARTS *(check all that apply)*

☒ Specification *Number of Pages* 14 ☐ CD(s), Number

☐ Drawing(s) *Number of Sheets* ☐ Other (specify)

☐ Application Data Sheet. See 37 CFR 1.76

METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT *(check one)*

☐ Applicant claims small entity status. See 37 CFR 1.27.

☒ A check or money order is enclosed to cover the filing fees

☐ The Commissioner is hereby authorized to charge filing fees or credit any overpayment to Deposit Account Number:

☐ Payment by credit card. Form PTO-2038 is attached.

FILING FEE AMOUNT ($) 160.00

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

☒ No.

☐ Yes, the name of the U.S. Government agency and the Government contract number are: _____.

Respectfully submitted,

SIGNATURE

Date 2/10/2004

REGISTRATION NO. 35,094
(if appropriate)

TYPED or PRINTED NAME John A. Castellano

Docket Number: 36731-000024/US

TELEPHONE 703-668-8000

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

XMAC - A New Fast Provable Secure MAC

No Author Given

No Institute Given

Abstract. We present a new fast proveable secure MAC called XMAC based on universal hashing. In the construction we use new families of universal hash functions which are especially well suited for tree-like hashing. Furthermore, we develop an effective tree-like hashing procedure. The proof of security is simple and easy to verify. The end result is a very effective MAC which is fast on both short and long messages achieving a peak performance of 2.2 clock cycles per byte on a Pentium III processor for a 64 bit tag with a forgery probability of 2^{-52}. The design makes it fast on most platforms.

Keywords: MAC, universal hash, tree, stream cipher, Rabbit

1 Introduction

A Message Authentication Code (MAC) provides a way to detect whether a message has been tampered with during transmission. The usual model for authentication includes three participants: a transmitter, a receiver and an opponent. The transmitter communicates a message over an insecure channel in which the opponent has the ability to introduce new messages as well as altering an existing message. Insertion of a new message by the opponent is called impersonation and modification of an existing message by the opponent is called substitution. In both cases the opponent's goal is to deceive the receiver into believing that the new message is authentic.

In many applications, it is of significant importance that the receiver can verify the integrity of a message. In some cases this is even more important than encryption [1]. Often encryption and authentication are both required. With the emergence of fast software-based encryption algorithms like Rijndael [2], SNOW [3], Rabbit [4] etc., the need for fast software-based message authentication codes is increasing. Some attempts have been made to construct an integrated MAC and encryption algorithm e.g. Helix [5]. However, such approaches make it hard to prove the security of the MAC part. Moreover, there exist constructions that can be proven secure with respect to an underlying cryptographic primitive. Prominent examples are HMAC [6] and the universal hashing approach [7].

The construction presented here is based on the universal hashing approach. Universal hashing was introduced in 1979 by Carter and Wegman [7]. A universal hash function family is a set of functions fulfilling certain combinatoric properties. For example, a family is called ϵ-universal if the probability of a collision in a randomly chosen function evaluated at two different points is no more than ϵ.

In 1981, Wegman and Carter suggested using universal hash functions for message authentication [8]. In their approach, a given message is hashed with a randomly chosen universal hash function whereafter the output is encrypted with a one-time-pad in order to obtain the MAC tag. Since the universal hash functions are only required to fulfill, in a cryptographical sense, a rather simple combinatoric property, they can usually be constructed to be very fast. Recent research has been successful in achieving impressive speeds. Noteable examples can be found in [9–13]. In particular, UMAC [12] which has been recommended in the NESSIE portfolio [14], has achieved speeds on a Pentium III processor of 1.8 clock cycles per byte[1] with forgery probability of 2^{-60} for a 64-bit tag. However, for very short messages the performance is slower, e.g. 12.6 clock cycles per byte for a message length of 43 bytes with the same forgery probability.

It is the aim of this paper to construct a Wegman-Carter based MAC which is fast on both short and long messages. The performance on short messages is important as the MAC function used in IPSec operates on 43-1500 bytes [15] and the MAC function used in TLS operates on 0-17 kilobytes. In addition, the setup procedure must be simple and fast, as the number of messages and amount of data processed per setup is small in many applications, e.g. TLS. Finally, the MAC is required to have verifiable-selectable assurance[2].

In order to achieve high performance, we introduce new families of universal hash functions especially well suited for tree-like hashing. These are obtained by reducing Δ-universal hash families to universal hash families. This results in significant performance gains for small compressions. Furthermore, we develop an effective tree-like hashing procedure which basically consists of combining a tree hash with a linear hash. The construction is provable secure (relative to a cryptographic primitive) with relatively simple proofs.

The paper is organized as follows. In section 2 we present the Definitions of the different classes of universal hash families and composition theorems. In section 3 we introduce a simple method to reduce delta-universal hash families to universal hash families. A modification to the simple tree hashing scheme is presented in section 4. Section 5 contains the specification of XMAC and the performance results are presented in section 6. We conclude in section 7.

2 Universal Hashing and Message Authentication

As mentioned above, Wegman and Carter [8] discovered that it is possible to use the notion of a randomly chosen strongly (see below) universal hash function to

[1] A 16-bit version of UMAC optimized for Pentium III SIMD technology has better performance. A similar modified version of XMAC using 16-bit multiplications could also gain a significant performance boost on the Pentium III processor.

[2] For a more detailed description of verifiable-selectable assurance, see [12]. In short, this means that the receiver can verify to lower assurance levels than for the full tag in order to increase performance.

compress a given message and encrypting it using a one-time-pad[3]. We describe briefly in the following why this is possible.

Let us first list well-known Definitions of universal hashing.

Definition 1 *An ϵ-almost-universal (ϵ-AU) family H of hash functions maps from a set A to a set B, such that for any distinct elements $x, x' \in A$:*

$$P_{h_k \in H}(h_k(x) = h_k(x')) \leq \epsilon \tag{1}$$

H is universal (U) if $\epsilon = 1/|B|$.

Universal hash families were first defined by Carter and Wegman in 1979 [7]. ϵ-AU hash families were defined by Stinson in 1991 [16].

Definition 2 *An ϵ-almost-Δ-universal (ϵ-AΔU) family H of hash functions maps from a set A to a set B, such that for any distinct elements $x, x' \in A$ and for all $a \in B$:*

$$P_{h_k \in H}(h_k(x) - h_k(x') = a) \leq \epsilon \tag{2}$$

H is Δ-universal (ΔU) if $\epsilon = 1/|B|$.

ϵ-AΔU is a generalization of the ϵ-Almost-Xor-Universal (ϵ-AXU) family of hash functions defined by Krawczyk in 1994 [17] to arbitrary abelian groups and was given by Stinson in 1996 [18].

Definition 3 *An ϵ-almost-strongly-universal (ϵ-ASU) family H of hash functions maps from a set A to a set B, such that for any distinct elements $x, x' \in A$ and all $a, b \in B$:*

$$P_{h_k \in H}(h_k(x) = a) = 1/|B| \tag{3}$$

and

$$P_{h_k \in H}(h_k(x) = a, h_k(x') = b) \leq \epsilon/|B| \tag{4}$$

H is strongly universal (SU) if $\epsilon = 1/|B|$.

SU hash families were first defined by Wegman and Carter in 1981 [8]. The concept of ϵ-ASU hash families was introduced in [8], and was later formalized by Stinson in 1991 [16].

Moreover, hash families can be combined in order to obtain new hash families. The below composition theorems (see [19]) describe what happens to the resulting ϵ, domains and ranges.

[3] Of course, a cryptographic primitive like a stream cipher can also be used to generate a pseudo-random key, but then the security depends on the security of the primitive.

Composition 1: If there exists an ϵ_1-AU family H_1 of hash functions from A to B and an ϵ_2-AU family H_2 of hash functions from B to C, then there exists an ϵ-AU family H of hash functions from A to C, where $H = H_1 \times H_2$, $|H| = |H_1| \cdot |H_2|$, and $\epsilon = \epsilon_1 + \epsilon_2 - \epsilon_1\epsilon_2 \leq \epsilon_1 + \epsilon_2$.

Composition 2: If there exists an ϵ_1-AU family H_1 of hash functions from A to B and an ϵ_2-ASU family H_2 of hash functions from B to C, then there exists an ϵ-ASU family H of hash functions from A to C, where $H = H_1 \times H_2$, $|H| = |H_1| \cdot |H_2|$, and $\epsilon = \epsilon_1 + \epsilon_2 - \epsilon_1\epsilon_2 \leq \epsilon_1 + \epsilon_2$.

From the Definitions it follows that strongly universal hashing can be used for message authentication. If we denote the probability for an impersonation attack to succeed by P_i and the probability for a substitution attack to succeed by P_s, we have the following Theorem(see for instance [8, 19, 20]):

Theorem 1 *There exists an ϵ-ASU family of hash functions from A to B if and only if there exists an authentication code with $|A|$ messages, $|B|$ authenticators and $k = |H|$ keys, such that $P_i = 1/|B|$ and $P_s \leq \epsilon$.*

A similar version for ϵ-AXU families has been proven by Krawczyk [17]. The particular Wegman-Carter MAC can be defined as:

Definition 4 *Given an ϵ-ASU family H of hash functions mapping from a set A to a set B, a nonce, n, and a random pad f(n), then the Wegman-Carter MAC is*

$$\mathrm{MAC_{WC}}(M; k, f(n)) = h_k(M) \oplus f(n), \tag{5}$$

where k is the random hash function key and M is the message.

A new nonce must be used for each application of the MAC to ensure the unconditional security of the construction

In the next section we will describe a method to reduce delta-universal hash functions to universal hash functions. It turns out that these new universal hash families are particularly well-suited for tree structures.

3 Reducing Delta-Universal Hash Functions to Universal Hash Functions

As seen above there are different classes of universal hash functions, e.g. strongly universal, almost strongly universal, delta-universal, almost delta-universal and so on. The latter are contained in the former. Furthermore, it is possible to convert classes into other classes. For example, it is possible to convert a delta-universal hash family into a strongly-universal hash family [11].

For our construction we convert the Δ-universal hash family, MMH^*, proposed by Halevi and Krawczyk [13] into a strongly universal hash family. This is accomplished by adding an additional key, k_{n+1}, in the following way:

$$MMH_K^{su}(M) = (\sum_{i=1}^{n} m_i k_i) + k_{n+1} \bmod p, \tag{6}$$

where p is a prime number, $M = m_1||...||m_n$ and $m_i, k_i \in \{0, ..., p-1\}$.

In some cases it is also beneficial to do the opposite, i.e. to reduce a stronger family into a weaker family. This is, of course, only relevant when a performance gain can be achieved. This is illustrated in the following.

Theorem 2 *Let H^Δ be an ϵ-almost-delta-universal hash family from a set A to a set B. Furthermore, consider an additional part of message $m_b \in B$. Then the family consisting of the functions $h_k(m, m_b) = h_k^\Delta(m) + m_b$ is ϵ-almost-universal for equal length messages.*

Proof. From the Definitions above we have for $m \neq m'$:

$$\Pr[h_k(m, m_b) - h_k(m', m_b') = 0] = \Pr[h_k^\Delta(m) + m_b - h_k^\Delta(m') - m_b' = 0] \quad (7)$$

or

$$\Pr[h_k^\Delta(m) - h_k^\Delta(m') = m_b' - m_b \equiv \delta] \leq \epsilon, \quad (8)$$

since H^Δ is an ϵ-almost-delta-universal family. The case when $m = m'$ but $m_b \neq m_b'$ is trivial ■

A very fast universal hash family is the NH family used in UMAC [12]. It was proposed in [21] based on a previous construction by Wegman and Carter:

$$NH_K(M) = \sum_{i=1}^{l/2} (k_{2i-1} +_w m_{2i-1}) \cdot (k_{2i} +_w m_{2i}) \bmod 2^{2w}, \quad (9)$$

where '$+_w$' means 'addition modulus 2^w', and $m_i, k_i \in \{0, ..., 2^w - 1\}$. It is an 2^{-w}-almost-delta-universal hash family. In [12] only the universal property is explicitly proven.

Corrolar 1 *The following version of NH:*

$$NH_K(m) = (k_1 +_w m_1) \cdot (k_2 +_w m_2) \bmod 2^{2w}, \quad (10)$$

is 2^{-w}-almost-Δ-universal for equal length messages.

Proof. This proof is just a slight modification of the one presented in [12]. We must show that

$$\Pr[(k_1 + m_1)(k_2 + m_2) - (k_1 + m_1')(k_2 + m_2') = \delta] \leq 2^{-w}, \quad (11)$$

as in [12] all arithmetics is carried out in $Z/2^{2w}$. We assume that $m_2 \neq m_2'$. Define $c = k_2 + m_2$ and $c' = k_2 + m_2'$. By assumption it follows that $c \neq c'$. So we have

$$\Pr[(k_1 + m_1)c - (k_1 + m_1')c' - \delta = 0] \leq 2^{-w}. \quad (12)$$

since from lemma 1 in [21] the equality will only be satisfied by one k_1 ■

According to Theorem2, this family can be reduced to an ϵ-almost-universal hash family:

$$F(m_1, m_2, m_3, m_4, k_1, k_2) = (m_1 +_{32} k_1)(m_2 +_{32} k_2) +_{64} m_3 +_{64} 2^{32} m_4, \quad (13)$$

where all arguments are 32-bit and the output is 64-bit. The collision bound is $\epsilon = 2^{-32}$.

The question is if this construction is useful and if so how can it be used most effectively? The construction is useful when the domain $|A|$ is not much larger than the range, $|B|$, but useless when $|A| \gg |B|$.

Thus, if relatively short messages are hashed for each key, the extra block results in a significant performance gain. This is the subject of the next section.

4 The Modified Tree Construction

An immediate use of the above defined hash family is in a tree-like construction. As an example of a tree construction we assume that the message length can be written as $|M| = b \cdot 2^n$ where b is some given block length. We also assume that a "two-to-one" universal hash family is given, i.e. a member of H takes a bitstring of length $2b$ and hashes it to a string of length b. Then we can construct a new universal hash family taking strings of the length $|M|$ and hashing it to strings of length b by the well-known tree-construction. The depth of the tree will be n. Clearly such a tree construction is the same as successively applying n parallel hashes. We define the parallel hash family [22] as follows:

Definition 5 *Given a message $M = m_1||...||m_{c^n}$ with length $|M| = bc^n$, we hash c blocks at a time with a universal hash function, $h_k \in H$ taking bc bits to b bits and concatenate the results. The result is a string with length bc^{n-1}. We denote the hash family by H^{par} and a member by h_k^{par}.*

$$h_k^{par}(M) = h_k(m_1, ..., m_c)||...||h_k(m_{c^n - c+1}, ..., m_{c^n}) \quad (14)$$

It is easy to see that if H has a collision bound of ϵ then so does the parallel hash, H^{par}. We define the usual tree construction as follows:

Definition 6 *Let a message M be given with length $|M| = bc^n$ for any integer n. We define a new hash family by applying $h_{k_i}^{par}$ n times, each time with a new random k_i. We denote the family by H_n^{tree} and a member by:*

$$h_{n;k}^{tree}(M) = h_{k_n}^{par} \circ h_{k_{n-1}}^{par} \circ ... \circ h_{k_1}^{par}(M). \quad (15)$$

We say that the tree has n levels.

Theorem 3 *The above defined family, H_n^{tree}, is a $1 - (1-\epsilon)^n$-universal family of hash functions for equal length messages.*

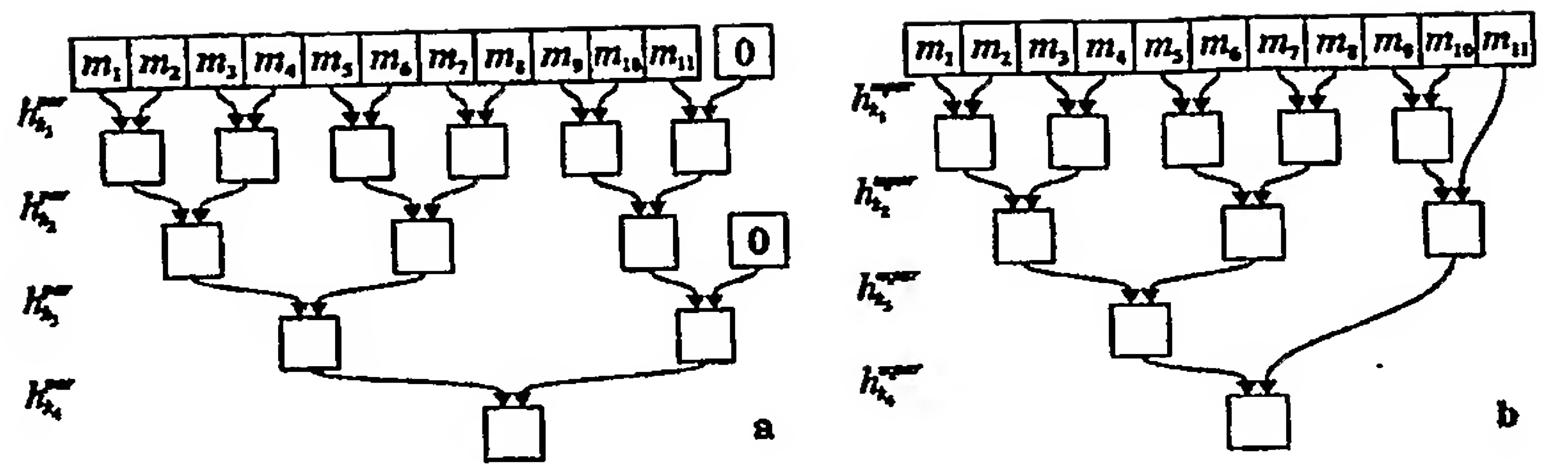

Fig. 1. Figure (a) illustrates the traditional tree construction using the parallel hash and figure (b) illustrates the modified tree construction using the modified parallel hash.

Proof. Let us define ϵ_i as the collision bound for $H^{tree}_{i;k'}$, then we have for $H^{tree}_{i+1;k''}$:

$$Pr[h^{par}_{k_{i+1}}(h^{tree}_{i;k'}(m)) - h^{par}_{k_{i+1}}(h^{tree}_{i+1;k'}(m')) = 0] \leq \epsilon_i(1-\epsilon) + \epsilon. \tag{16}$$

Solving the recurrence we get:

$$Pr[h^{par}_{k_n}(h^{tree}_{n-1;k}(m)) - h^{par}_{k_n}(h^{tree}_{n-1;k}(m')) = 0] \leq$$
$$(1-\epsilon)^{n-1}\epsilon + \epsilon\sum_{i=1}^{n-2}(1-\epsilon)^i + \epsilon = 1-(1-\epsilon)^n < n\epsilon \quad \blacksquare \tag{17}$$

The hash family defined by Eq. 13 is very well suited for binary tree constructions. However, in such a tree the message lengths must be the block length times a power of two. An immediate generalization would be to do as suggested by Wegman and Carter [8]. They suggest breaking the message into substrings of length $2b$ and if necessary pad the last substring with zeroes. The resulting string is hashed with the parallel hash. If necessary, the resulting string is again padded with zeroes. This is repeated until the resulting string has length b.

This procedure is not always optimal as illustrated in Fig. 1a. The reason being that for most message lengths, e.g. message lengths not equal to a power of two, extra applications of the universal hash function are needed. Of course, this is only significant for short messages. We propose the construction below.

First we define a modified parallel hash:

Definition 7 *Given a universal hash family, H, taking bc bits to b bits with members, h_k, consider the message M, where $|M|$ is a multiple of the block size b, i.e. $M = m_1||..||m_q$ and $|M| = qb$. Define $r_c \equiv q \bmod c$, then the modified parallel hash can be defined as:*

$$h^{mpar}_k(M) = \begin{cases} h_k(m_1,...,m_c)||...||h_k(m_{q-c+1},...,m_q) \text{ if } r_c = 0 \\ h_k(m_1,...,m_c)||...||h_k(m_{q-c-r_c+1},...,m_{q-r_c})||m_{q-r_c+1}||..||m_q \text{ if } r_c \neq 0 \end{cases} \tag{18}$$

Property 1 *The modified parallel hash is ϵ-almost universal on equal length messages.*

Proof. In the first case, where q is a multiple of c we simply have a parallel hash and the bound on the collision probability is ϵ. In the case where q is not a multiple of c, there are two possible situations. Either the difference in the messages M and M' is in the part, which is processed by h_k, or in the part which is not processed, but simply concatenated to the result. In the first situation the bound on the collision probability is ϵ. In the second situation the collision probability is trivially zero. Therefore, the bound for the collision probability of the modified parallel hash is $\epsilon_{mpar} = \max(\epsilon, 0) = \epsilon$ ■

It is straightforward to define a modified tree hash, i.e. define it as in Definition 6 but use the modified parallel hash instead of the usual parallel hash.

Corrolar 2 *Given a message with length $|M| = b(c^{n-1} + r)$ where $0 \leq r < c^n - c^{n-1}$ the modified tree hash defines a $1 - (1 - \epsilon)^n$-almost universal family of hash functions on equal length messages.*

Proof. This follows from Theorem3, when the usual parallel hash is replaced by the modified parallel hash, since both are ϵ-almost universal, and that the number of levels are the same in both cases[4]■

As an example consider the case when $c = 2$. The message is divided into blocks of size b. If the message length is not a multiple of b, zeros are appended to the message such that the length becomes a multiple of b. If the length hereafter is a multiple of $2b$, the hash function is applied to each block and the results are concatenated. If the length is an odd multiple of b, the hash function is applied to each block except the last block. The results and the last block are concatenated. The procedure is repeated until the size of the result is b. The construction is illustrated in Fig. 1b.

Note that the construction can alternatively be defined in the following way (we use the binary case as an example): Let the message length be given by $|M| \equiv b\sum_{i=0}^{n} a_i 2^i$ where $a_i \in \{0, 1\}$. To each term, $a_i 2^i$, in the sum, corresponds to a tree with i levels. We order these trees according to size with the largest tree first. More precisely, we use the tree hash for each group of data corresponding to a term in the sum, concatenate the result, and linearly hash it backwards, i.e. take the b-bit block as output from the last tree and hash it with the result of the second to last tree and so on, until only one b-bit string is left. In other words, the construction consists of a series of concatenated tree hashes followed by a linear hash [22]. For the example in Fig. 1b the message length can be written as: $|M| = b(2^3 + 2^1 + 2^0)$. There is one tree with 3 levels, one with 1 level

[4] In a Wegman-Carter binary tree hash, a message consisting of an odd number of blocks is padded up such that the number of blocks is even. This is done after each application of the parallel hash. The number of levels is equal to the number of levels of a message of the nearest larger power of two. Now it is easy to convince oneself that the number of levels of the modified tree hash is exactly the same.

and one with 0 levels. The hash results of those trees are then linearly hashed starting with the result from the smallest tree.

The above construction is only almost-universal for equal length messages. To ensure universality for different length messages we simply concatenate the length of the given message in a fixed z-bit format [12, 22]:

Definition 8 *Fix $z > 0$ and let the message, M, have any length less than 2^z. Define $L_z = |M|$ to be the z-bit representation of the length and define H^* as the family:*

$$h_k^*(M) = h_k(M) || L_z. \tag{19}$$

We then have the following property:

Property 2 *The hash function family H^* is $1 - (1 - \epsilon)^n$-almost universal.*

Proof. In the case $|M| \neq |M'|$, the collision probability is trivially zero. In the case $|M| = |M'|$, the collision probability is defined according to Corrolar 2 by the number of levels necessary to compress the message ■

In order to use our universal hash function in a MAC, according to Theorem1 we need to apply a strongly universal hash function to the output of h_k^*, i.e. $h_{k'}^{SU}(h_k^*(M))$. This strongly universal hash function maps an input of size $b + z$ bits to an output of a size appropriate for the collision probability.

Theorem 4 *The hash function family consisting of the members $h_{k'}^{SU}(h_k^*(M))$ is $\epsilon_{tree} + (1 - \epsilon_{tree})\epsilon_{SU}$-almost-strongly universal.*

Proof. According to composition 2. ■

Finally, it is easily seen that the amount of key material, $N_{\text{MAC}}(M)$, needed for a given message M is given by

$$N_{\text{MAC}}(M) = N_U ceil[\log_c(|M|/(b))] + N_{SU}, \tag{20}$$

where N_U is the amount of key material needed for the basic almost-universal hash function in the tree and N_{SU} is the amount of key material needed for the strongly-universal hash function used in the end. Note that the amount of needed key material is the same as for the usual tree MAC.

In the next section we explicitly specify XMAC.

5 The XMAC Specification

A schematic pseudo-code of XMAC is presented in the box below. Below we shortly describe the different steps of the algorithm.

To generate the key material for use in the universal hash functions, any secure pseudo-random generator is applicable. In this algorithm we will use the stream cipher Rabbit [4], which is seeded with a 128-bit key. We define the

maximal length of a message to be 2^{64} bits, requiring maximally 58 levels in the tree and, hence, 58 64-bit keys. Furthermore, 6 keys belonging to the interval $\{0, ..., 2^{31}-2\}$ need to be generated for the strongly universal hash function.

To process the message, it is divided into 64-bit blocks, and padded with zeroes if necessary. The message is then processed with the hash function defined in eq. (13), $h(k, m_1, m_2)$, where $k, m_1, m_2 \in \{0, ..., 2^{64}-1\}$, in a modified binary tree construction as defined in Definition 7 and the text below, until the message is compressed to 64 bits. The length of the message measured in bits is represented as a 64-bit number and concatenated to the 64-bit result.

The resulting 128-bit block is appended with 22 zeroes and divided into five 30-bit blocks and hashed with the strongly universal version of the hash family MMH* defined in Eq. 6. The prime field is chosen to be $\{0, ..., 2^{31}-2\}$ to ensure simple overflow handling and modulus calculation in 32-bit implementations.

The final tag is generated by XOR'ing the output of the hash function with a pseudorandom pad, according to Definition 4. To generate the pseudo-random pad we use the IV-setup function of Rabbit with a nonce as initialization vector. The size of the output from the hash function is in principle 31 bits. However, we encrypt the output with a pseudorandom pad of size 32 bits, to make the tag match the 32-bit register size.

Function $h(k, m_1, m_2)$

1. return $(m_1 +_{32} k) \cdot ((m_1 \gg 32) +_{32} (k \gg 32)) +_{64} m_2$

Function $XMAC_K(M, nonce)$

1. Generate 64-bit keys: $Rabbit_K = k_1 || ... || k_{58}$
2. Generate keys $\in \{0, ..., 2^{31}-2\}$: $Rabbit_K = k_1^{su} || ... || k_6^{su}$
3. $L = |M|$
4. while $|M| \bmod 64 \neq 0$ do: $M = M||0$
5. for $i = 1$ to $i = ceil[log_2(L/64)]$ do:
$$M = \begin{cases} \text{if } t \text{ is even, return } h(k_i, m_1, m_2)||...||h(k_i, m_{t-1}, m_t) \\ \text{if } t \text{ is odd, return } h(k_i, m_1, m_2)||...||h(k_i, m_{t-2}, m_{t-1})||m_t \end{cases}$$
6. Append to M the 64-bit message length L, $Q = M||L$
7. Divide Q into 30-bit blocks and append 22 zeroes, $Q = s_1||...||s_5||0||...||0$
8. $S = (\sum_{i=1}^{5} s_i k_i^{su}) + k_6^{su} \bmod 2^{31}-1$,
9. return $S \oplus Rabbit_K(nonce)$

The forgery probability depends on the number of levels in the tree, the bound on the collision probability for each level in the tree is $\epsilon_h = 2^{-32}$. For the strongly universal hash family we have $\epsilon_{SU} = 1/(2^{31}-1) \approx 2^{-31}$. The maximal number of levels in the tree for a 2^{64} bit message is 58. Using composition 2 and Theorem 3 the forgery probability is: $\epsilon \leq \epsilon_{SU} + (1-\epsilon_{SU})(1-(1-\epsilon_h)^n) \approx 2^{-31} + (1-2^{-31})(1-(1-2^{-32})^{58}) \approx 2^{-26.09}$.

A forgery probability of $2^{-26.09}$ is insufficient for most applications. However, a simple method to reduce the forgery probability is to hash the message y times with independent keys and concatenate the results. This method results in a forgery probability of $2^{-26.09y}$. To obtain 128-bit security we need to hash the message 5 times yielding a probability of 2^{-130} and a tag size of 160 bits. In particular, this leads to the verifiable-selective assurance as each 32-bit tag can be verified independently.

6 Performance

We measured the performance of the algorithm specified above on a 1000 MHz Pentium III processor. The speed-optimized version was programmed in assembly language inlined in C and compiled using the Intel C++ 7.0 compiler. All performance results in this section are based on generating a $2 \cdot 32$ bit tag.

In Table 1 the performance results are presented for two cases. The first case is where the pseudo-random pad is generated by the Rabbit stream cipher after being re-initialized by the IV Setup function, with the nonce as initialization vector. The second case is where the pseudo-random pad is generated by continuing the extraction of pseudo-random data from the Rabbit stream cipher. This eliminates the need to perform the rather expensive IV-setup, but is only useful when messages are guaranteed to be received in the same order as generated. This situation corresponds to interpreting the nonce as an iteration number of the stream cipher. However, in most applications the IV-setup is necessary, as for example in IPSec communication.

Since, the key material in XMAC depends on the length of the message, optimized versions can be used in applications where the message length is upper bounded. For example, in typical IPSec applications, the message length cannot exceed 1500 bytes and when authenticating TLS [23] protected data, each message cannot exceed 17 kilobytes. Furthermore, the strongly universal hash function is simplified since parts of the input is zero, see eq. (6). The properties of XMAC when the message length is limited is shown in Table 2.

The performance of XMAC and UMAC is illustrated in Fig. 6. Without the IV-setup XMAC is about a factor of 4 faster than UMAC for very shor messages and with the IV-setup the performance on short messages is about the same. On long message the speed is still remarkable, and almost the same as UMAC.

Table 1. Performance results with and without IV-setup. "Key setup" includes generating all keys for the ϵ-AU and SU hash functions, "Universal hash" includes processing the tree, and "Finalization" includes the SU hash function and generating the pseudo-random pad.

Function	IV-setup	Without IV-setup
Key setup	4372 cycles	4372 cycles
Universal hash	2.2 cycles/byte	2.2 cycles/byte
Finalization	500 cycles	150 cycles

Table 2. XMAC properties for various limited message lengths. "Memory req." denotes the amount of memory required to store the internal state including key material, temporary results and an instance of the Rabbit stream cipher. "Fin. no IV" denotes finalization without IV-setup and "Fin. IV" denotes finalization with IV-setup.

Max message size	Forgery pr.	Memory req.	Setup	Fin. No IV	Fin. IV
2^{11} bytes (e.g. IPSec)	2^{-57}	364 bytes	1108 cycles	126 cycles	476 cycles
2^{15} bytes (e.g. TLS)	2^{-56}	492 bytes	1364 cycles	126 cycles	476 cycles
2^{32} bytes	2^{-54}	1044 bytes	2484 cycles	138 cycles	488 cycles
2^{61} bytes	2^{-52}	1980 bytes	4372 cycles	150 cycles	500 cycles

However, the forgery probabilities are a little better for UMAC. Note also that XMAC is still not fully optimized.

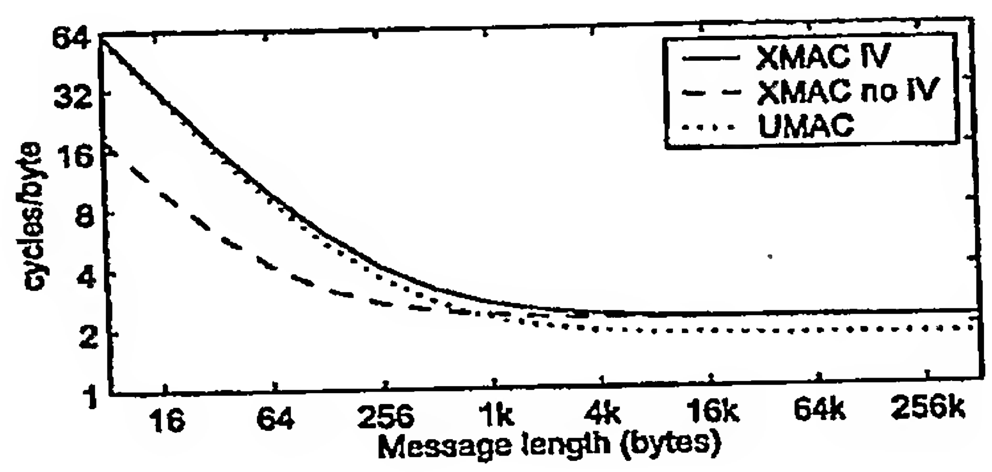

Fig. 2. The performance of UMAC and XMAC as a function of message length.

7 Conclusions

We presented a MAC called XMAC bas ed on universal hashing. We introduced new families of universal hash functions especially well suited for tree-like hashing. These were obtained by reducing Δ-universal hash families to universal hash families. Furthermore, we developed an effective tree-like hashing procedure which consists basically of combining a tree hash with a linear hash. The proof of security is simple and easily verifiable. XMAC is both fast on short and long messages achieving a peak performance of 2.2 cycles per byte on a Pentium III processor with a forgery probability of 2^{-52} for a 64-bit tag. The necessary key material for the hash functions is only 976 bytes, making the setup very fast. The design makes it fast on most platforms, and especially well suited for small 32-bit processors, due to the small memory requirements.

8 Claims

1. A method for generating a cryptographically secure checksum (also called MAC or tag) of a digital message to be used for authenticating the message. The method comprising dividing the message into blocks of a certain size, which are combined using a compression method to obtain fewer blocks.

2. A method according to claim 1 where the process of compression is repeated a number of times so as to end up with 1 or more blocks being the checksum or to be used for calculating the checksum.

3. A method according to claim 1 and 2 where the compression method used to compress two input blocks (m_1 and m_2) into one output block ($h(k, m_1, m_2)$) given a cryptographic key (k) is

$$h(k, m_1, m_2) = (m_1 +_{32} k) \cdot ((m_1 \gg 32) +_{32} (k \gg 32)) +_{64} m_2 \qquad (21)$$

References

1. N. Ferguson and B. Schneier: *Practical Cryptography*, Wiley (2003)
2. J. Daemen and V. Rijmen: *AES proposal: Rijndael*, http: //www.nist.gov/aes, (1998)
3. P. Ekdahl and T. Johansson: *A New Version of the Stream Cipher SNOW*, Proceedings of Selected Areas in Cryptography 2002, Springer, pp. 49-61 (2002)
4. M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen and O. Scavenius: *Rabbit: A New High-Performance Stream Cipher*, Proceedings of Fast Software Encryption (FSE) 2003, Springer, (2003)
5. N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks, and T. Kohno: *Helix Fast Encryption and Authentication in a Single Cryptographic Primitive* , Proceedings of Fast Software Encryption (FSE) 2003, Springer, (2003)
6. H. Krawczyk, M. Bellare and R. Canetti: *HMAC: Keyed-Hashing for Message Authentication*, RFC 2104, (1997)
7. J. L. Carter and M. N. Wegman: *Universal Classes of Hash Functions*, J. Computer and System Sciences 18, pp. 143-154 (1979)
8. M. N. Wegman and J. L. Carter: *New Hash Functions and their Use in Authentication and Set Equality*, J. Computer and System Sciences 22, pp. 265-279, (1981)
9. P. Rogaway: *Bucket Hashing and its Application to Fast Message Authentication*, Proc. CRYPTO 95, LNCS 963, pp. 29-42, Springer-Verlag, (1995)
10. J. Bierbrauer, T. Johansson, G. Kabatianskii and B. Smeets: *On Families of Hash Functions via Geometric Codes and Concatenation*, Proc. CRYPTO'93, LNCS 773, pp. 331-342, Springer-Verlag, (1994)
11. M. Etzel, S. Patel and Z. Ramzan: *Square Hash: Fast Message Authentication Via Optimized Universal Hash Functions*
12. T. D. Krovetz: *Software-Optimized Universal Hashing and Message Authentication*, Ph. D. Thesis, http://www.cryptonessie.org (2000)
13. S. Halevi and H. Krawczyk: *MMH: Software Message Authentication in the Gbit/second Rates*, Proc. 4th Workshop on Fast Software Encryption, LNCS vol. 1267, pp. 172-189, Springer, (1997)

14. NESSIE consortium: *Portfolio of recommended cryptographic primitives*, http://www.cryptonessie.org
15. T. D. Krovetz: *Software-Optimized Universal Hashing and Message Authentication*, chapter 3, Ph. D. Thesis, http://www.cryptonessie.org (2000)
16. D. R. Stinson: *Universal Hashing and Authentication Codes*, "Advances in Cryptology - CRYPTO '91", Lecture Notes in Computer Science 576, pp. 74-85, (1992)
17. H. Krawczyk: *LFSR-Based Hashing and Authentication*, "Advances in Cryptology - CRYPTO '94", Lecture Notes in Computer Science 839, pp. 129-139, (1994)
18. D. R. Stinson: *On the connections between universal hashing, combinatorial designs and error-correcting codes*, Congressus Numerantium 114, pp. 7-27, (1996)
19. D. R. Stinson, *Universal Hashing and Authentication Codes*, Designs, Codes and Cryptography 4, pp. 369-380, (1994)
20. W. Nevelsteen and B. Preneel: *Software Performance of Universal Hash Functions*, EUROCRYPT99, LNCS 1592, pp. 24-41, Springer-Verlag, (1999)
21. J. Black, S. Halevi, H. Krawczyk, T. Krovetz and P. Rogaway: *UMAC: Fast and secure message authentication*,, Proceedings of CRYPTO'99, Springer-Verlag LNCS 1666, pp.216-233, (1999)
22. M. Bellare and P. Rogaway: *Collision-Resistant Hashing: Towards Making UOWHFs Practical*, Advances in Cryptology Crypto 97 Proceedings, LNCS Vol. 1294, Springer-Verlag, (1997)
23. T. Dierks, and C. Allen: *The TLS Protocol Version 1.0*, RFC 2246, (1999)
24. UMAC website: http://www.cs.ucdavis.edu/~rogaway/umac/